ELSEVIER Theoretical Computer Science 287 (2002) 251–265

Theoretical Computer Science

www.elsevier.com/locate/tcs

Spiking neurons and the induction of finite state machines ☆

Thomas Natschläger*, Wolfgang Maass

Institute for Theoretical Computer Science, Technische Universität Graz, Inffeldgasse 16b, A-8010 Graz, Austria

Abstract

We discuss in this short survey article some current mathematical models from neurophysiology for the computational units of biological neural systems: neurons and synapses. These models are contrasted with the computational units of common artificial neural network models, which reflect the state of knowledge in neurophysiology 50 years ago. We discuss the problem of carrying out computations in circuits consisting of biologically realistic computational units, focusing on the biologically particularly relevant case of computations on time series. Finite state machines are frequently used in computer science as models for computations on time series. One may argue that these models provide a reasonable common conceptual basis for analyzing computations in computers and biological neural systems, although the emphasis in biological neural systems is shifted more towards asynchronous computation on analog time series. In the second half of this article some new computer experiments and theoretical results are discussed, which address the question whether a biological neural system can, in principle, learn to behave like a given simple finite state machine. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Computational neuroscience; Spiking neurons; Dynamic synapses; Finite state machines; Grammatical inference

1. Introduction

Computational models for neural systems have often concentrated on the processing of static stimuli. However, numerous ecologically relevant signals have a rich temporal

☆ This work was supported by the project P12153 of the Fonds zur Förderung wissenschaftlicher Forschung, and the NeuroCOLT project of the EC.

* Corresponding author.

E-mail addresses: tnatschl@igi.tu-graz.ac.at (T. Natschläger), maass@igi.tu-graz.ac.at (W. Maass).

URLs: http://www.igi.TUGraz.at/igi/tnatschl, http://www.igi.TUGraz.at/igi/maass

PII: S0304-3975(02)00099-3

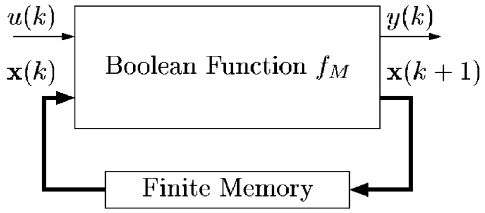

Fig. 1. Implementation of a FSM as sequential machine (as defined in [13]). The boolean function f_M relates the current input symbol $u(k)$ and the current state $\mathbf{x}(k)$ to the output symbol $y(k)$ and the next state $\mathbf{x}(k+1)$. We assume in this article that $u(k)$, $\mathbf{x}(k)$ and $y(k)$ are encoded as bit strings.

structure, and neural circuits must process these signals in real time. In many signal processing tasks, such as audition, almost all of the information is embedded in the temporal structure. In the visual domain, movement represents one of the fundamental features extracted by the nervous system. Hence, it is not surprising that in the last few years there has been increasing interest in the dynamic aspects of neural processing. Processing of real-world time-varying stimuli is a difficult problem, and represents an unsolved challenge for artificial models of neural functions. Simultaneously, in computer science several areas such as, for example, computer vision, robotics, and machine learning have also increased their efforts to deal with dynamic real-world inputs.

A computational model which is, in principle, suitable for analyzing computations on time series is that of a *finite state machine* (FSM). Informally speaking, a FSM is an abstraction of a device that operates on strings of symbols. But in contrast to the closely related deterministic finite automaton (DFA), which just accepts or rejects an input, the FSM generates for each input string a corresponding output string in an online manner. The mapping is not arbitrary, but is constrained by the finite number of internal states the FSM is allowed to use (for a formal definition see [13]). A FSM can be implemented in several ways. One particular implementation of a FSM which naturally associates time with the FSM is the so-called *sequential machine* (SM). It consists of a boolean function f_M which relates the current input symbol $u(k)$ and the current internal state $\mathbf{x}(k)$ to the current output symbol $y(k)$ and the next internal state $\mathbf{x}(k+1)$; see Fig. 1. The concept of a FSM is a very general model for computation on time series. However, FSMs are abstract devices and there is no direct relationship between a FSM and most common models for biological neural systems.

A model for computations in biological neural systems that captures the essential aspects of biological neural systems has to take into account the way how biological neurons transmit and process information (see Section 2 for more details). The output of a biological neuron consists of a sequence of almost identical electrical pulses, or “*spikes*” (see Fig. 2). These so-called *spike trains* are the time series which are processed by a biological neural system.

In this survey article we point to some new results that address the question what computations a network of spiking neurons can perform. In particular, we will show that a network of spiking neurons cannot only perform all computations that a certain subclass of FSMs can perform, but can also *learn* to do so.

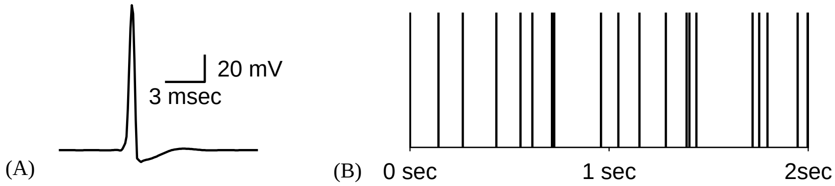

Fig. 2. (A) Typical action potential (spike). (B) A typical spike train produced by a neuron (each firing time marked by a bar).

This article is organized as follows: In Section 2 we describe a formal model of a network of spiking neurons. In Section 3 we discuss a computational model which is suitable for analyzing computations on spike trains. Based on this computational model we present empirical data from computer simulations that show how a subclass of FSMs can be learned with a network of spiking neurons. We also briefly outline the theory behind these experiments (Section 4).

2. A model for a network of spiking neurons

In this section, we discuss a formal model of spiking neurons that captures the fact that biological neurons communicate via spikes.[1] For an excellent survey about modeling neural systems we refer to [5].

If one ignores all dynamic aspects, then a spiking neuron has some similarity to the familiar threshold gate in computer science (see for example [23]). A threshold gate outputs 1 if and only if the weighted sum of its inputs reaches some threshold. Similarly a spiking neuron i "fires", i.e. generates a short electrical pulse, which is called *action potential*, or "*spike*" (see Fig. 2(A)), if the current input at time t drives the membrane potential $h_i(t)$ above some threshold θ_i. Each such spike has the same shape. Hence the output of a spiking neuron is a sequence of spikes at certain points in time, informally called "*spike train*" (see Fig. 2(B)). Formally, the spike train generated by neuron i is simply the set of firing times $F_i \subset \mathbb{R}^+$ ($\mathbb{R}^+ = \{x \in \mathbb{R}\colon\ x \geqslant 0\}$).

In the simplest (deterministic) model of a spiking neuron one assumes that a neuron i fires whenever the membrane potential h_i (which models the electric membrane potential at the "trigger zone" of neuron i) reaches the threshold θ_i. h_i is the sum of the so-called excitatory postsynaptic potentials (EPSPs) and inhibitory postsynaptic potentials (IPSPs), which result from the firing of "presynaptic" neurons j that are connected through a "synapse" to neuron i (see Fig. 3).

The firing of a neuron j at time $\hat{t}$ contributes to the potential $h_i(t)$ at time t an amount that is modeled by the term $w_{ij}(\hat{t})\varepsilon_{ij}(t-\hat{t})$, which consists of the *synaptic strength* $w_{ij}(\hat{t}) \geqslant 0$ and a *response-function* $\varepsilon_{ij}(t-\hat{t})$. Biologically, realistic shapes of such response functions are indicated in Fig. 3(B). If Γ_i is the set of all neurons

[1] The "spike trains" demo software which illustrates information processing with spikes can be downloaded from `http://www.cis.TUGraz.at/igi/tnatschl/spike_trains_eng.html`.

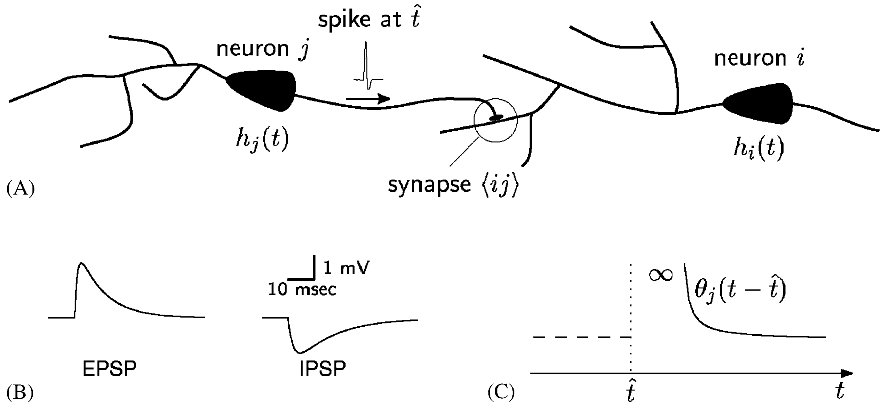

Fig. 3. Information processing with spikes. (A) Spikes are generated by a threshold process whenever the membrane potential $h_j(t)$ crosses the threshold $\theta_j(t-\hat{t})$. The spike travels down the axon of neuron j. Via the synaptic connection $\langle ij \rangle$ the spike is transformed into a postsynaptic response at neuron i. (B) Typical shape of a postsynaptic response, which is either positive (EPSP) or negative (IPSP), of a biological neuron. (C) Typical shape of the threshold function $\theta_j(t-\hat{t})$ of a biological neuron ($\hat{t}$ is the time of its most recent firing).

presynaptic to neuron i, then the membrane potential $h_i(t)$ at the trigger zone of neuron i at time t is given in terms of the sets F_j of firing times of these presynaptic neurons j by

$$h_i(t) := \sum_{j \in \Gamma_i} \sum_{\hat{t} \in F_j : \hat{t} < t} w_{ij}(\hat{t}) \varepsilon_{ij}(t-\hat{t}) \,. \tag{1}$$

The membrane potential $h_i(t)$ does not really correspond to the weighted sum of a threshold gate since it varies over time. Unfortunately, not even the threshold θ_i is static. If a neuron i has fired at time $\hat{t}$, it will not fire again for a few ms after $\hat{t}$, no matter how large its current potential $h_i(t)$ is (*absolute refractory period*). Then for a few further ms it is still reluctant to fire, i.e. a firing requires a larger value of $h_i(t)$ than usual (*relative refractory period*). Both of these refractory effects are modeled by a suitable *threshold function* $\theta_i(t-\hat{t})$, where $\hat{t}$ is the time of the most recent firing of i. A typical shape of the function $\theta_i(t-\hat{t})$ for a biological neuron is indicated in Fig. 3(C).

The synaptic strength $w_{ij}(\hat{t})$ can be interpreted as the amplitude of the postsynaptic response triggered by the firing of neuron j at time $\hat{t}$, whereas the time course and the sign (EPSP or IPSP) of the response are determined by the response function $\varepsilon_{ij}(t-\hat{t})$.[2]

[2] The restriction of $w_{ij}(\cdot)$ to non-negative values—in combination with positive (EPSP) or negative (IPSP) response functions $\varepsilon_{ij}(\cdot)$—is motivated by the empirical result that a biological synapse is either excitatory or inhibitory, and that it does not change its sign in the course of a learning process. In addition, for most biological neurons j, either all response-functions $\varepsilon_{ij}(\cdot)$ for postsynaptic neurons i are excitatory (i.e. positive), or all of them are inhibitory (i.e. negative).

In most mathematical models for spiking neurons one ignores the time dependence of the factor $w_{ij}(\cdot)$, and views it as a constant w_{ij} which only changes on the slow time scale of learning. Thus w_{ij} corresponds to a "weight" in traditional models for artificial neural nets. However, a large number of experimental studies have shown that biological synapses have an inherent short-term dynamics, which controls how the pattern of amplitudes of postsynaptic responses depends on the temporal pattern of the incoming spike train [17,25]; see Fig. 6 for an example. Various quantitative models have been proposed [1,19] involving a small number of hidden parameters, that allow us to predict the response of a given synapse to a given spike train once proper values for these hidden synaptic parameters have been found. In this article we will use the model of Markram et al. [19] and point to one possible role of such *dynamic synapses* (see Section 3).

We assume that for some specified subset of *input neurons* their firing times (spike trains) are given from the outside as *input* to the network. The firing times for all other neurons are determined by the previously described rules, and the output of the network is given in the form of the spike trains for the neurons in a specified set of *output neurons* (see Fig. 7 for an example).

We would like to point out that the formal model for a spiking neuron that we have outlined so far is a coarse simplification. In particular, the membrane potential $h_i(t)$ at the trigger zone of a neuron i is, in general, not a linear sum of incoming pulses. Both *sublinear* and *superlinear* summation occur in biological neurons. Also the threshold function θ_i (see Fig. 3(C)) varies from neuron to neuron. For example, in the case of periodically firing neurons ("pacemaker" neurons) the threshold function θ_i may also rise again after its initial decline. With regard to further details about biological neural systems we refer to [10–12,21,27].

3. Induction of finite state machines

In this section, we will sketch a new computational model for analyzing computations on spike trains and show on the basis of this model how, in principle, a network of spiking neurons can be trained to mimic the behavior of a given definite memory machine (DMM) which is a special case of a FSM. This approach is quite different from previous theoretical work [14] where it is shown that certain recurrent networks of spiking neurons can, in principle, simulate any given Turing machine. The results reported in [14] are based on a construction where the current state of the simulated Turing machine is stored by the activation of a proper set of oscillators consisting of a small network of spiking neurons. No such—biologically somewhat dubious—oscillators are required in the feedforward network that we will investigate in this article.

We will consider the problem of constructing a network $\mathcal{N}$ of spiking neurons which approximates the behavior of a given DMM M with binary input and output symbols, i.e. M transforms an input string $\underline{u} \in \{0,1\}^*$ to an output string $\underline{y} \in \{0,1\}^*$. The current output $y(k)$ of a DMM depends only on a finite number d of previous inputs $u(k-1), \ldots, u(k-d)$ in addition to the current input $u(k)$ (see Fig. 4; for a

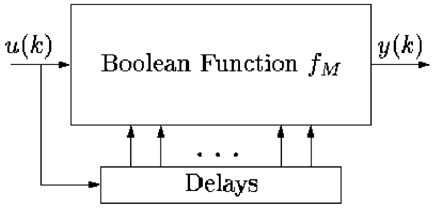

Fig. 4. Implementations of a DMM as sequential machine. The current output $y(k)$ is determined by some boolean function f_M, which is applied to a finite number d of previous inputs $u(k-1),\ldots,u(k-d)$ in addition to the current input $u(k)$.

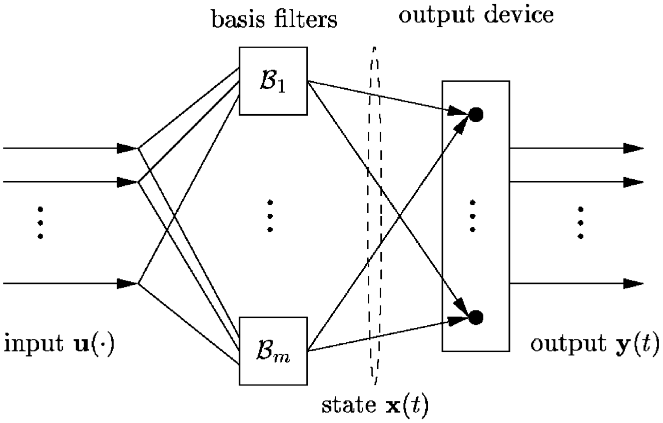

Fig. 5. A new computational model for analyzing computations on time series $\mathbf{u}(\cdot)$, such as for example spike trains.

formal definition of a DMM see [13]).[3] We will denote by $\mathscr{D}_d$ the class of all DMMs M for which $y(k)$ can be written in the form

$$y(k) = f_M(u(k), u(k-1), u(k-2), \ldots, u(k-d)). \qquad (2)$$

Note that there are $2^{2^{d+1}}$ choices for the function f_M, or equivalently $|\mathscr{D}_d| = 2^{2^{d+1}}$.

3.1. The network architecture

On an abstract level the computational model that we will consider, see Fig. 5, consists of an array $\mathscr{B}_1 \ldots, \mathscr{B}_m$ of subsystems—denoted as *basis filters* in the following—whose collective outputs define a continuous state vector $\mathbf{x}(\cdot)$. Note that this state vector $\mathbf{x}(\cdot)$ differs in general from the current state of the simulated DMM. It is more closely related to the state vector of a dynamical system [24]. This state $\mathbf{x}(\cdot)$ is transformed into an output $\mathbf{y}(\cdot)$ by means of a suitable memoryless output device.

[3] We will write $u(k)$ ($y(k)$) for the kth symbol (bit) of the string $\underline{u}$ ($\underline{y}$).

3.1.1. The set of basis filters

From a theoretical point of view it would suffice to choose the basis filters $\mathscr{B}_1 \ldots, \mathscr{B}_m$ as a proper set of delay lines. In this case the resulting network is structurally similar to the implementation of a DMM as sequential machine in the sense that the output is computed directly from the input and delayed versions of it via the output device (which corresponds to the boolean function f_{M} in the case of a DMM). It is not surprising that such an architecture can simulate an arbitrary DMM M if the output device is able to compute the equivalent of the boolean function f_{M}. However, from a biological point of view such an implementation is not satisfactory since transmission delays of more than 30 ms are rarely found in cortical circuits [22]. On the other hand biologically organisms need to respond appropriately to time series that are spread out over much larger time spans. For example, the temporal distance between saccades (rapid eye movements) is on average around 300 ms.

Hence, we propose a biologically more plausible implementation for the set of basis filters: we employ a multitude of biologically realistic *dynamic synapses* with a uniform delay, but with a suitable distribution of the parameters which control the current synaptic strength. We employ the model of Markram et al. [19], where three parameters U_{ij}, F_{ij}, and D_{ij} control the dynamics of a synaptic connection $\langle ij \rangle$ between neurons j and i. A fourth parameter A_{ij}—which corresponds to the synaptic "weight" in static synapse models—scales the absolute sizes of the postsynaptic responses. The resulting model predicts the amplitude $w_{ij}(t_m) = A_{ij} \cdot u_{ij}(t_m) \cdot R_{ij}(t_m)$ of the postsynaptic response to the $(m+1)$th spike in a spike train $F_j = \{t_0, t_1, \ldots, t_m\}$ with interspike intervals (ISIs) $\Delta_k = t_{k+1} - t_k, k = 0, \ldots, m-1$, in terms of two internal dynamic variables u_{ij} and R_{ij} ranging over $[0,1]$, whose dynamics is governed by the following recursive equations [4]

$$\begin{aligned} u_{ij}(t_{k+1}) &= U_{ij} + u_{ij}(t_k)(1 - U_{ij}) \exp(-\Delta_k / F_{ij}), \\ R_{ij}(t_{k+1}) &= 1 + (R_{ij}(t_k) - u_{ij}(t_k) R_{ij}(t_k) - 1) \exp(-\Delta_k / D_{ij}) \end{aligned} \tag{3}$$

with the initial conditions $u(t_0) = U$ and $R(t_0) = 1$ for the first spike.

It is reported in [9] that the synaptic parameters U, F, D are quite heterogeneous, even within a single neural circuit (see Fig. 6(A)). Note that the time constants D and F are in the range of a few hundred ms. The synapses investigated in [9] can be grouped into three major classes: facilitating (F1), depressing (F2) and recovering (F3). Fig. 6(B) compares the output of a typical F1-type and a typical F2-type synapse in response to a typical irregular spike train. One can see that the same input spike train yields markedly different outputs at these two synapses.

Whenever we talk about synaptic strength or weight in the following we refer to the parameter A_{ij}. In our learning experiments we just modify this parameter A_{ij}, whereas U_{ij}, F_{ij}, D_{ij} are assumed to be fixed (at some biologically realistic values).

The vector of internal states $\mathbf{x}(t) = \langle x_1(t), \ldots, x_m(t) \rangle$ of our computational model consists of the individual outputs $x_j(t)$ of the m basis filters. In our concrete

[4] To be precise: the term $u_{ij}(t_k)R_{ij}(t_k)$ in Eq. (3) was erroneously replaced by $u_{ij}(t_{k+1})R_{ij}(t_k)$ in the corresponding Eq. (2) of Markram et al. [19]. The model that they actually fitted to their data is the model considered in this article.

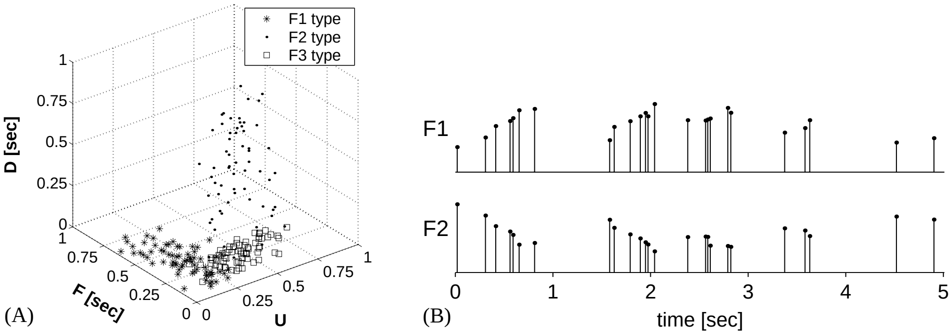

Fig. 6. Synaptic heterogeneity. (A) The parameters U, D, and F can be determined for biological synapses. Shown is the distribution of values for inhibitory synapses investigated in [9] which can be grouped into three mayor classes: facilitating (F1), depressing (F2) and recovering (F3). (B) Synapses produce quite different outputs for the same input for different values of the parameters U, D, and F. Shown are the amplitudes $w(t_k)$ (height of vertical bar) of the postsynaptic response of a F1-type and a F2-type synapse to an irregular spike train (horizontal positions of vertical bars mark the spike times). Parameters: $U=0.16$, $D=45$ ms, $F=376$ ms for F1, and $U=0.25$, $D=706$ ms, $F=21$ ms for F2.

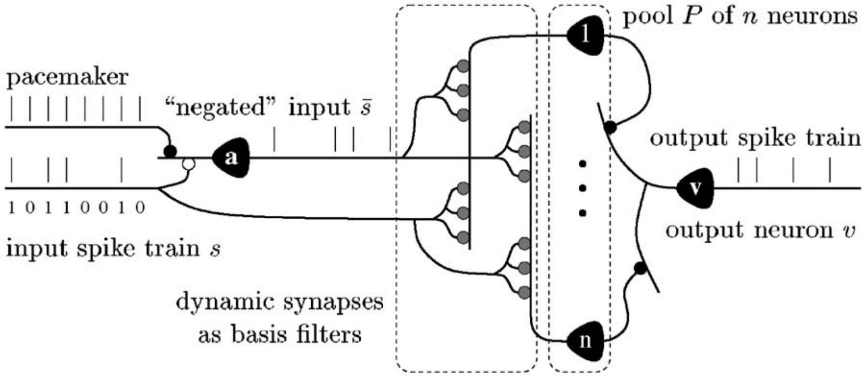

Fig. 7. A network of spiking neurons which can learn to mimic a given definite memory machines. Via its excitatory (●) and inhibitory (○) synapse the neuron a generates a spike train ("negated" input) which corresponds to the bitwise inverted input string. During learning only the strengths of the dynamic synapses (gray circles) to the n neurons in pool P are adjusted.

implementation the jth basis filter is a single dynamic synapse which either receives the input spike train s or the "negated" input spike train $\bar{s}$ as input; see Fig. 7. Hence, we define $x_j(t_k) := u_j(t_k) \cdot R_j(t_k)$ where j refers to the jth dynamic synapse. Later we will determine the synaptic strength A_{ij} through a suitable learning algorithm.

3.1.2. The output device

Here we consider as output device a small pool P of spiking neurons and view the fraction $p(t)$ of neurons in P which fire during a short time interval $[t-\Delta, t]$ as the output of the computation. To convert $p(t)$ into an output spike train one could use a spiking neuron v which receives input from all neurons in pool P (see Fig. 7). In that case the time course of the membrane potential $h_v(t)$ would approximate very closely the time course of $p(t)$. The spiking mechanism of neuron v translates $h_v(t)$ into an output spike train. For sake of simplicity, we assume that this implementation of the output device has no temporal delays. However, more realistically one could assume that the output device maps $\mathbf{x}(t)$ to $p(t+\Delta)$ and $h(t+\Delta+\Delta')$ for some fixed delays Δ, Δ' in the range of 5–10 ms.

Recent results [2,15] show that such a single population P of spiking neurons (even without lateral or recurrent connections) has enough computational power so that a large class of functions that map state vectors $\mathbf{x}(t)$ to the population activity $p(t)$ can be computed. Furthermore, there exists a new learning algorithm [2], which allows to tune the synaptic strengths A_{ij} of neurons $i \in P$ such that a given function g from $\mathbf{x}(t)$ to $p(t)$ is computed. From a biological point of view it is particularly appealing that this algorithm requires for each neuron just the application of a perceptron-like local learning rule: Whenever too many neurons in P are firing the weights A_{ij} of all neurons $i \in P$ which fire are changed such that the membrane potential $h_i(t)$ decreases for the current input $\mathbf{x}(t)$. In the case where too few neurons fire the weights of all neurons which are not firing are changed such that the membrane potential increases for the given input (for details see [2]).

To summarize, the proposed network for implementing a given DMM M is a feed-forward network of spiking neurons consisting of an array of dynamic synapses (the basis filters) which supplies the input $\mathbf{x}(t)$ to a pool P of spiking neurons (the output device). All information about preceding inputs is carried within these dynamic synapses. The pool P in turn drives the neuron v which produces the desired output spike train; see Fig. 7.

3.2. *The training procedure*

3.2.1. *The learning task*

The goal of the training experiments which we are now going to describe is to show that it is possible to mimic the behavior of a DMM M which is chosen randomly from the class $\mathscr{D}_d$ defined at the beginning of Section 3 with a *fixed network architecture* (see Fig. 7). Only the values A_{ij} of the synaptic strengths of the dynamic synapses are different when the network of Fig. 7 is used to simulate two different DMMs $M, M' \in \mathscr{D}_d$. Note that the number of dynamic synapses, the number n of neurons in pool P, and the values of the parameters U_{ij}, D_{ij} and F_{ij} are equal for all target DMMs $M \in \mathscr{D}_d$. Hence, to simulate a given $M \in \mathscr{D}_d$ we have to find proper values A_{ij} for the synaptic strengths of the dynamic synapses. We will use the learning algorithm from Auer et al. [2] to accomplish this task. In order to do so we have to generate a suitable set of training examples.

Speaking more formally, to simulate a given $M \in \mathscr{D}_d$ by a network $\mathscr{N}$ of spiking neurons (see Fig. 7) we need to implement a certain function $g_M : \mathbb{R}^m \to [0,1]$—the *target function*—which maps state vectors $\mathbf{x}(t)$ to population activity $p(t)$. The network $\mathscr{N}$ implements the function $g_{\mathscr{N}} : \mathbb{R}^m \to [0,1]$ which is defined by the values $A_{ij} \in \mathbb{R}, i = 1 \ldots n, j = 1 \ldots m$ of the synaptic strengths of the dynamic synapses. Hence, the task of the learning algorithm is to find proper values A_{ij} such that $g_{\mathcal{N}}$ approximates g_M.

3.2.2. Encoding of input and output

In order to simulate a given DMM M with a network of spiking neurons we have to choose a particular encoding for the input and output of the DMM M in terms of an input and output spike train. We represent a value $u(k)=1$ ($y(k)=1$) by the occurrence of a spike and a value $u(k)=0$ ($y(k)=0$) by no spike in the corresponding time interval. Since spiking neurons normally operate in an asynchronous way one has to define times t_k when one should "look for a spike". To accomplish this we use an extra input ("pacemaker") to our network of spiking neurons which supplies the times t_k in the form of a regular spike train of 40 Hz (see Fig. 7). In a biological context such invariant periodic input is for example provided by well-known global rhythms that show up in EEG-recordings; such as rhythmic discharge in the gamma frequency band (20–70 Hz).

3.2.3. Generation of training examples

As a consequence of our encoding scheme it is only relevant for the times t_k how a state vector $\mathbf{x}(t_k)$ is mapped to the population activity $p(t_k)$ ($p(t_k)=g_{\mathcal{N}}(\mathbf{x}(t_k))$). If the network $\mathcal{N}$ receives an input spike train which encodes a string $\underline{u}\in\{0,1\}^q$ and should produce an output spike train which encodes the string $\underline{y}\in\{0,1\}^q$—the output of the DMM M—then $p(t_k)=g_{\mathcal{N}}(\mathbf{x}(t_k))$ should be as close as possible to $y(k)=g_M(\mathbf{x}(t_k))$ for all $t_1,\dots,t_q$. Therefore, the set of training examples consists of tuples $\langle\mathbf{z},o\rangle$ where $\mathbf{z}=\mathbf{x}(t_k)$ and $o=y(k)$ is the target output. A simulation of such network $\mathcal{N}$ with the time points t_k marked by dashed lines in shown in Fig. 8.

After the learning algorithm has found proper values for the synaptic strengths A_{ij} either a large ($y(k)=1$) or a small ($y(k)=0$) fraction of neurons $i\in P$ will fire around time t_k. Therefore if one chooses a proper threshold for the output neuron v it will ($y(k)=1$) or will not ($y(k)=0$) fire around time t_k according to the value of $y(k)$.

3.2.4. Results

Some results of our learning experiments[5] for $d=3$ ($|\mathcal{D}_3|=2^{2^4}=65536$) are summarized in Fig. 9. We have randomly chosen 650 DMMs $M\in\mathcal{D}_3$ from the uniform distribution over all $M\in\mathcal{D}_3$. For each of these DMMs we trained a network $\mathcal{N}$ with the learning algorithm from Auer et al. [2] on a single randomly chosen string $\underline{u}\in\{0,1\}^*$ of length 400. To check whether the network has really learned to mimic the DMM at hand we measured the performance (percent of correct output spikes) of the trained network on a randomly chosen test string $\underline{\tilde{u}}\in\{0,1\}^*$ of length 400. As one can see

[5] For numerical simulations of spiking neurons we used the so called leaky integrate-and-fire neuron model (see e.g. [5]) with a membrane time constant of $\tau_m=20$ ms. The only parameters that are modified in our learning experiments are the synaptic parameters A_{ij} that scale the $m=18$ components of $\mathbf{x}(t)$ (i.e. the output $u_j(t)\cdot R_j(t)$ of the jth dynamic synapse) individually for each neuron i in the population P of size 200. Each neuron is connected to the input spike train s and to the "negated" input spike train $\bar{s}$ via nine synapses (postsynaptic current time constant $\tau_s=1$ ms). Each set of nine synapses consists of three depressing, three facilitating and three recovering synapses with time constants D and F in the range of 30–100 ms.

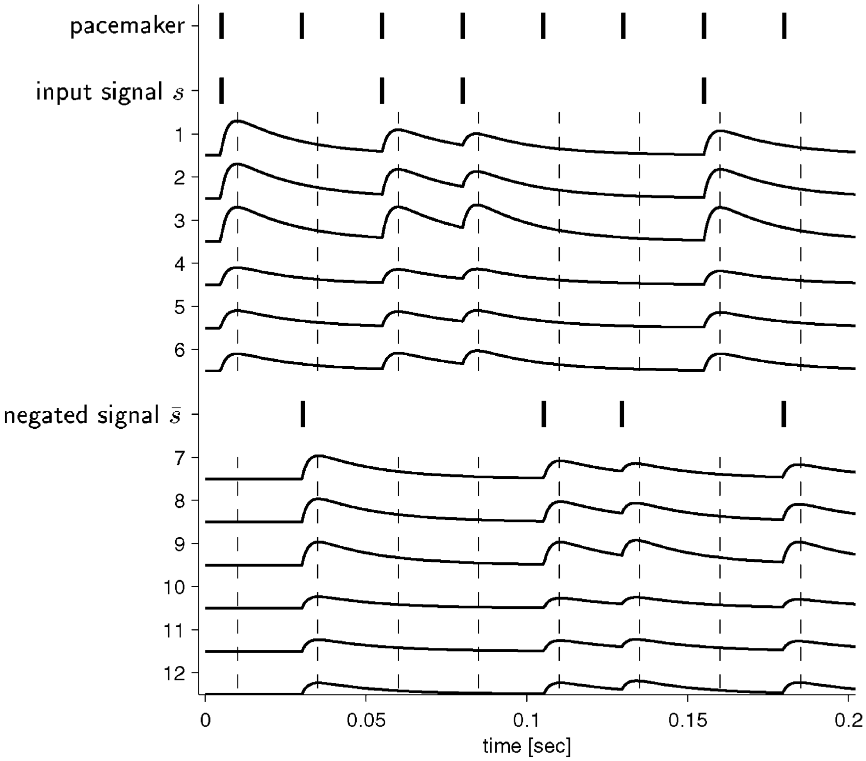

Fig. 8. Numerical simulation of the network of Fig. 7 for the input string $\underline{u} = 10110010$. The traces 1–12 show the contributions to the membrane potential of selected dynamic synapses of the network. Synapses 1–6 receive the direct input s whereas synapses 7–12 get the negated signal $\bar{s}$ as input. If one looks for example closely at trace 1 one can see that the amplitudes of these EPSPs differ from spike to spike. These subtle differences, which also vary from synapse to synapse, can serve as short-term memory for a neural system, since the amplitude for the current spike contains information about the preceding part of the spike train. The dashed lines mark the times t_k at which the state vector $\mathbf{x}(t_k)$ is recorded in order to serve—in combination with the target output $y(k)$—as training example for our learning experiments. Note the time difference (a few ms) between the pacemaker and the time points t_k. We have chosen the times t_k in that way because usually a few ms elapse until a EPSP or IPSP reaches its extremal value.

in Fig. 9 the performance on the test string $\underline{\tilde{u}}$ is near 100% (average 96.07%) for all 650 randomly chosen DMMs. This indicates that at network of spiking neurons (see Fig. 7) can learn to behave like an arbitrary DMM $M \in \mathscr{D}_3$.

4. Theoretical background

In this section, we briefly review the theory which underlies the computational model employed in Section 3. Most of this theory was developed in the context of common models for analog computations in populations of spiking neurons (“population coding”

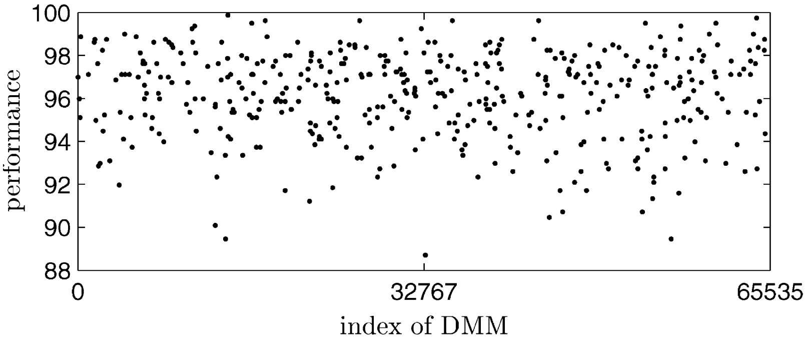

Fig. 9. Networks of spiking neurons can mimic the behavior of DMMs $M \in \mathscr{D}_3$. We plotted the performance (percent of correct output spikes) of the trained network for a random test string $\tilde{\underline{u}} \in \{0,1\}^{400}$ for 650 DMMs $M \in \mathscr{D}_3$ where the boolean function f_{M} was chosen randomly. Each DMM $M \in \mathscr{D}_3$ can be assigned a unique index (plotted on the X-axis) by interpreting the string which consists of the enumeration of all possible outputs of f_{M} as a binary number.

or "space rate coding"), see [4,6]. In the following, we refer to such systems which map a vector of time varying input signals $\mathbf{u}(\cdot)$ onto another vector of time varying output signals $\mathbf{y}(\cdot)$ as *filter*. We adopt the common notation $(\mathscr{F}\mathbf{u})(t)$ to denote the output that the filter $\mathscr{F}$ gives at time t for the input function $\mathbf{u}$.

4.1. A universal approximation theorem for filters

In the preceding section, we had presented empirical evidence for the approximation capabilities of our network model for computations in the time series (spike train) domain. This gives rise to the question, what the theoretical limits of their approximation capabilities are. The result presented in [16] (and its extension to spike trains in [18]) shows that basically there are no significant a priori limits. Furthermore, in spite of the rather complicated system of equations that defines our network model, one can give a precise mathematical characterization of the class of filters that can be approximated by them. This characterization involves the following basic concepts.

An arbitrary filter $\mathscr{F}$ is called *time invariant* if a shift of the input functions by a constant t_0 just causes a shift of the output function by the same constant t_0. Another essential property of filters is *fading memory*. A filter $\mathscr{F}$ has fading memory if and only if the value of $(\mathscr{F}\mathbf{u})(0)$ can be approximated arbitrarily closely by the value of $(\mathscr{F}\mathbf{v})(0)$ for functions $\mathbf{v}$ that approximate the functions $\mathbf{u}$ for sufficiently long bounded intervals $[-T,0]$ sufficiently well. Informally speaking, this means that the output of a filter with fading memory primarily depends on inputs within a certain time interval, i.e. it has essentially finite memory. Hence a time invariant filter with fading memory is a generalization of a DMM in the setting of analog filters $\mathscr{F}$. The class of time invariant filters with fading memory arguably includes practically any biologically relevant filter.

In [16] the computational power of feedforward networks with dynamic synapses where the computational units are pools of neurons is analyzed. Their rigorous theoretical result states that even with just a single hidden layer such networks can approximate a surprisingly large class of nonlinear filters: all *time invariant* filters with *fading memory*. The proof of this result relies on the Stone–Weierstrass Theorem, and is contained as the proof of Theorem 3.4 in [16]. Furthermore, it is empirically shown in [20] that gradient descent suffices to approximate a given (quadratic) filter by a model for a rather small neural system with dynamic synapses. It is also demonstrated that the simple feedforward network with biologically realistic synaptic dynamics yields nonlinear filter performance comparable to that of an artificial network model proposed in [3] that was designed to yield good performance in the time series domain without any claims of biological realism.

Very recently the techniques used in [16] have been extended to cover also the case of *spike trains* $\mathbf{u}(\cdot)$ as input time series [18]. This new result implies that a very rich class of maps $\mathscr{F}$ from input spike trains $\mathbf{u}(\cdot)$ to output spike trains $\mathbf{y}(\cdot)$ can be implemented by circuits of spiking neurons with the basic architecture shown in Fig. 5.

4.2. A new universal approximation theorem for static functions

There also exists a rigorous proof [2,15] that a single population P of threshold gates (or spiking neurons) can approximate any given continuous function on any given compact set. This provides another component of the theoretical basis for the learning approach in Section 3, where only output devices of this type are considered.

5. Discussion

We have sketched in this short survey article mathematical models for biological neurons and synapses that reflect the current state of knowledge in neurophysiology. It becomes clear that neither biological neurons nor biological synapses are modeled well by the "neurons" and "synapses" of common artificial neural network models, which ignore the inherent temporal dynamics of their biological counterparts. Furthermore, we have indicated new approaches towards implementing specific finite state machines through circuits consisting of these more realistic dynamic computational units.

We would like to argue that computations on time series provide a better paradigm for neural computation than the computations on static inputs that are more frequently investigated in the context of artificial neural networks. One simple model for computations on time series, the finite state machine (FSM), plays an important role both in neuroscience and in theoretical computer science. We have shown that a subclass of such FSMs can be implemented quite well by circuits consisting of biologically realistic models for neurons and synapses, in a way which differs strongly from the common implementations of such machines in boolean circuits or models for artificial neural networks [7,8,26].

In contrast to most previous work on implementations of FSMs in neural networks we have not just shown that specific machines of this type, but randomly selected machines from a large class of FSMs can be implemented, even with the same architecture of the neural circuit. Furthermore we have shown that such fixed neural circuit can in principle learn to simulate a randomly chosen simple FSM.

References

[1] L.F. Abbott, J.A. Varela, K.A. Sen, S.B. Nelson, Synaptic depression and cortical gain control, Science 275 (1997) 220–224.

[2] P. Auer, H. Burgsteiner, W. Maass, The p-delta learning rule for parallel perceptrons, submitted for publication, 2001.

[3] A.D. Back, A.C. Tsoi, A simplified gradient algorithm for IIR synapse multilayer perceptrons, Neural Comput. 5 (1993) 456–462.

[4] A.P. Georgopoulos, A.P. Schwartz, R.E. Ketner, Neuronal population coding of movement direction, Science 233 (1986) 1416–1419.

[5] W. Gerstner, Spiking neurons, in: W. Maass, C. Bishop (Eds.), Pulsed Neural Networks, MIT Press, Cambridge, 1999, pp. 3–53.

[6] W. Gerstner, Populations of spiking neurons, in: W. Maass, C. Bishop (Eds.), Pulsed Neural Networks, MIT Press, Cambridge, MA, 1999, pp. 261–293.

[7] C.L. Giles, B.G. Horne, T. Lin, Learning a class of large finite state machines with a recurrent neural network, Neural Networks 8 (9) (1995) 1359–1365.

[8] C.L. Giles, C.B. Miller, D. Chen, H.H. Chen, G.Z. Sun, Y.C. Lee, Learning and extracting finite state automata with second-order recurrent neural networks, Neural Comput. 4 (1992) 393–405.

[9] A. Gupta, Y. Wang, H. Markram, Organizing principles for a diversity of GABAergic interneurons and synapses in the neocortex, Science 287 (2000) 273–278.

[10] D. Johnston, S.M.S. Wu, Foundations of Cellular Neurophysiology, MIT Press, Cambridge, MA, 1995.

[11] C. Koch, Biophysics of Computation: Information Processing in Single Neurons, Oxford University Press, Oxford, 1999.

[12] C. Koch, I. Segev, Methods in Neural Modeling: From Ions to Networks, MIT Press, Cambridge, MA, 1998.

[13] Z. Kohavi, Switching and Finite Automata Theory, McGraw-Hill, New York, 1978.

[14] W. Maass, Lower bounds for the computational power of networks of spiking neurons, Neural Comput. 8(1) (1996) 1–40, electronically available via http://www.cis.TUGraz.at/igi/maass/psfiles/75.ps.gz.

[15] W. Maass, On the computational power of winner-take-all, Neural Comput. 12(11) (2000) 2519–2536, electronically available via http://www.cis.TUGraz.at/igi/maass/psfiles/113j.ps.gz.

[16] W. Maass, E.D. Sontag, Neural systems as nonlinear filters, Neural Comput. 12 (8) (2000) 1743–1772, electronically available via http://www.cis.TUGraz.at/igi/maass/psfiles/107rev.ps.gz.

[17] H. Markram, M. Tsodyks, Redistribution of synaptic efficacy between neocortical pyramidal neurons, Nature 382 (1996) 807–810.

[18] W. Maass, T. Natschläger, H. Markram, Real-time computing without stable states: A new framework for neural computation based on perturbations, submitted for publication, 2001.

[19] H. Markram, Y. Wang, M. Tsodyks, Differential signaling via the same axon of neocortical pyramidal neurons, Proc. Nat. Acad. Sci. 95 (1998) 5323–5328.

[20] T. Natschläger, W. Maass, A. Zador, Efficient temporal processing with biologically realistic dynamic synapses, Network: Computation in Neural Systems 12 (2001) 75–87.

[21] F. Rieke, D. Warland, R. de Ruyter van Steveninck, W. Bialek, Spikes: Exploring the Neural Code, MIT Press, Cambridge, MA, 1997.

[22] G. Shepherd, The Synaptic Organization of the Brain, 3rd Edition, Oxford University Press, Oxford, 1995.

[23] K.-Y. Siu, V. Roychowdhury, T. Kailath, Discrete Neural Computation: A Theoretical Foundation, Information and System Sciences Series, Prentice-Hall, Englewood Cliffs, NJ, 1995.

[24] S.H. Strogatz, Nonlinear Dynamics and Chaos: With Applications in Physics, Biology, Chemistry, and Engineering (Studies in Nonlinearity), Addison-Wesley, Reading, MA, 1994.

[25] J.A. Varela, K. Sen, J. Gibson, J. Fost, L.F. Abbott, S.B. Nelson, A quantitative description of short-term plasticity at excitatory synapses in layer 2/3 of rat primary visual cortex, J. Neurosci. 17 (1997) 220–224.

[26] R.L. Watrous, G.M. Kuhn, Induction of finite-state languages using second order recurrent networks, Neural Comput. 4 (1992) 406–414.

[27] A.M. Zador, The basic unit of computation, Nature Neurosci. 3 (Suppl.) (2000) 1167.